Optimization opportunities in GRAPHITE

Tobias Grosser

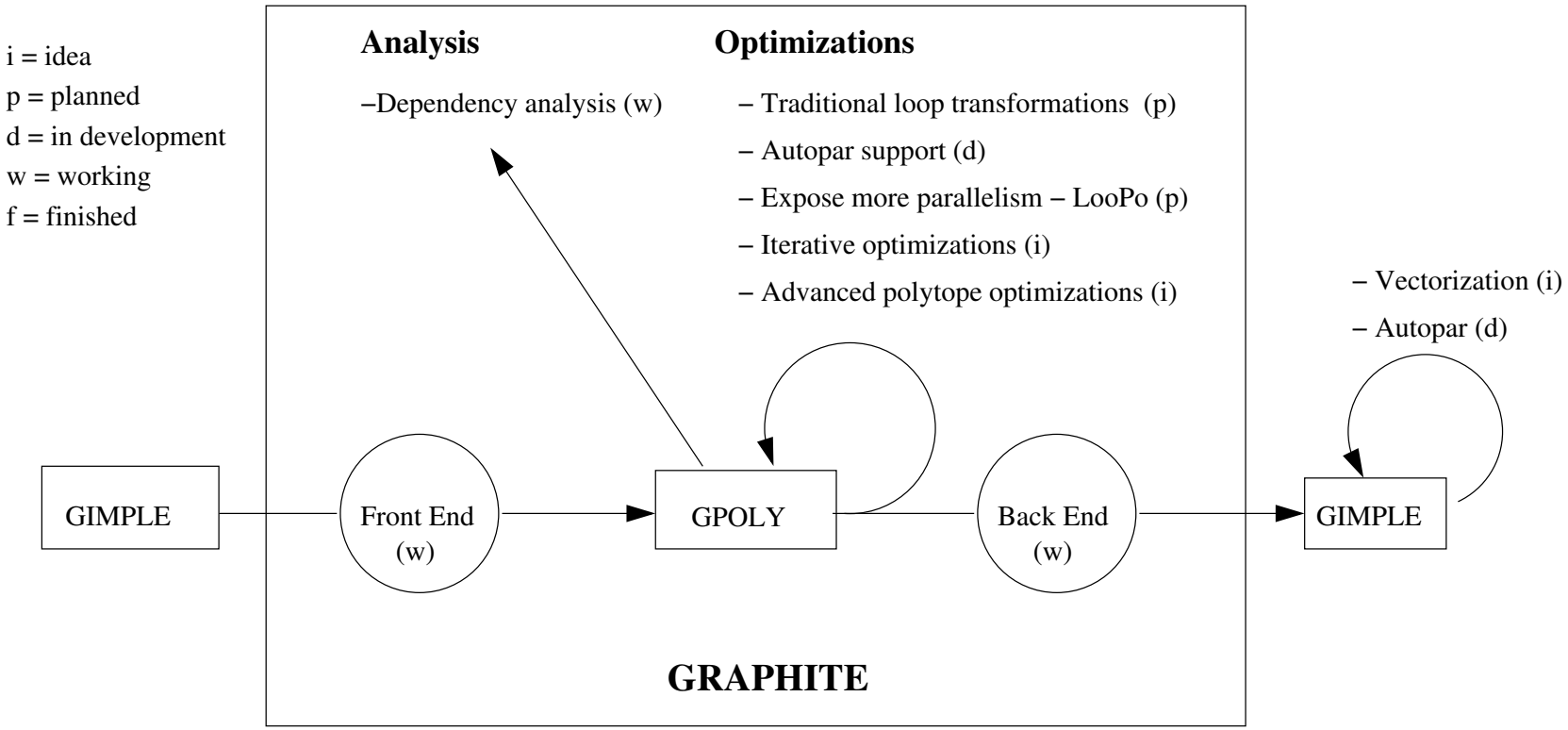
University of Passau

June 8, 2009

Agenda

1. Status of GRAPHITE optimizations
2. The polytope model
   - The polytope model = Array SSA?
   - What code can be represented?
   - GPOLY - The polytope representation in GRAPHITE
3. Coverage of GRAPHITE
   - Compile time and hot spot coverage
   - How to extend coverage

Status of GRAPHITE



Arrays as set of scalar values

$$f = \{A \rightarrow 0, B \rightarrow 1, c \rightarrow 2\}$$

```
A [5][8] = 10;
A [5][8] = c;

for (i = 0; i<100; i++) {
  B[3][2] = A[5][8]
  for (j = 0; j<50; j++)
    A[3][4] = B[3][2]
  B[5][8] = A[3][4]
}
```

Figure: Non SSA code

```
(0,5,8) = 10;
(0,5,8) = (2,0,0);

for (i = 0; i<100; i++) {
  (1,3,2) = (0,5,8);
  for (j = 0; j<50; j++)
    (0,3,4) = (1,3,2);
  (1,5,8) = (0,3,4);
}
```

Figure: Constant array accesses based on memory cells

Introduce SSA on the memory cells

```
(0,5,8) = 10; //S1
(0,5,8) = (S0,(2,0,0)); //S2

for (i = 0; i < 100; i++) {
  (1,3,2) = (S2,(0,5,8)); //S3.1
  for (j = 0; j < 50; j++)
    (0,3,4) = (S3.1,(1,3,2)); //S3.2.1
  (1,5,8) = (S3.2.1,(0,3,4)); //S3.3
}
```

Figure: Constant array SSA using the statement schedule

```
void foo (int k)
  A[5] = 10;
  A[k] = c;

  for (i=0; i<100; i++)
    B[5i+k] = A[5]

  for (j=0; j<100; j++)
    ... = B[j]
```

```
void foo (int k)
  (0,5) = 10; // S1
  (0,k) = (S0,(2,0)); // S2

  for (i=0; i<100; i++)
    (1,5i+k) = (S2,(0,5)); // S3.1

  for (j=0; j<100; j++)
    ... = (S3.1 (j=5i+k) && S0 ((j-k)%5!=0),(1,j))); // S4.1
```

Polytopemodel > SSA?

Properties that have both:

- Only one def for every use
- Explicit use def chains

Additional features of the polytope model

- More abstract
- No PHI nodes required
- Able to statically analyze the complete SCoP

Price to pay

- More expensive in memory and compile time
- Not general, but limited to certain code regions

What code can be represented?

- Structured code
- Affine loop bounds
- Constant loop strides
- Conditions contain comparisions ($<, <=, >, >=, ==, !=$) between affine functions
- Array accesses affine

Analysis is working on GIMPLE, so the textual representation does not matter $\rightarrow$ hand made goto based loops work as well.

GPOLY

SCoP The optimization unit
$scop = (< bb >)$

Black Box A operation, where only the memory accesses are known
$bb = (domain, scattering, < dr >)$

iteration domain Set of statement iterations
$0 <= i <= 100,\ 5 <= j <= 2 + k$

scattering matrix The execution order of statement iterations, as the lexicographic order of t0, t1, ..., tn
$t0 = 0, t1 = i, t2 = j$

data reference The memory cells accessed. "a" is the alias set, "sn" are array dimensions
$a = 1; s0 = 12, s1 = i, s2 = 3j + k$

Possible optimizations in GPOLY

- iteration domain
  - Remove statement iterations
- scattering matrix
  - Change the execution order of statement iterations to improve cache locality
  - ... expose parallelism (for vectorizer or autopar)
- data reference (Not yet supported)
  - Change the data layout to improve cache behaviour
  - ... to save memory.
  - Add additional memory to allow more parallelism

Coverage of GRAPHITE

Is it worth to write optimizations using GRAPHITE?

Compile time coverage

- Coverage analysis as in gcov
- Count GIMPLE stmts, loop header and conditions
- Items are covered, if they are part of an interesting SCoP

Pro

- System independent
- Code generation support not necessary

Con

- Influenced by changes in other passes
- Does not show runtime

Hot spot coverage

- One reference run with **-fprofile-arcs -ftest-coverage**
- Scale number of stmts, loops and conditions by execution count (obtained with **-fprofile-use**)

Pro

- Hot spots are taken into account
- Reproducable and exact measurement

Con

- Does still not show runtime

```
a = 10;
for (i=0; i<100; i++){
  b = 2 * i;
  if (b >= 0)
    c = 3;
}
```

	Compile	Hot Spot
loops	1/1	100/100
conds	2/2	200/200
stmts	7/9	700/702

```
bb0:
  a = 10;
  i = 0;

loop_head:
  b = 2 * i;
  if (b >= 0) goto then
  else goto exit_cond

then:
  c = 3;
  goto exit_cond

exit_cond:
  i++;
  if (i < 100) goto loop_head;
```

Code coverage GRAPHITE branch + reductions patch

	Compile time			Hotspot		
Benchmark	**loops**	**conds**	**stmts**	**loops**	**conds**	**stmts**
410: bwaves	3.45	1.35	1.46	0.19	0.16	0.09
436: cactusADM	**2.50**	**0.96**	**3.54**	**81.83**	**68.83**	**99.41**
454: calculix	**10.68**	**4.10**	**5.28**	**61.79**	**55.66**	**76.09**
435: gromacs	9.08	3.49	2.77	7.75	2.94	2.20
464: h264ref	**6.24**	**2.05**	**2.21**	**20.42**	**12.58**	**7.96**
470: lbm	**19.23**	**7.35**	**35.79**	**0.07**	**0.02**	**0.02**
437: leslie3d	5.76	2.49	1.23	4.36	4.34	0.72
481: wrf	**18.57**	**6.36**	**7.70**	**34.45**	**29.72**	**54.86**
434: zeusmp	3.41	1.03	1.11	0.84	0.43	0.11
others (14)			< 1			< 1
average of all 23	**4.66**	**1.52**	**2.90**	**9.45**	**7.68**	**10.62**

Figure: Coverage May 2009 [in %]

How to improve coverage

- GRAPHITE SCoP detection
  - Allow more conditions $if(2i < 0 || 3i > 1)$
  - Allow unknown access functions $A[rand()]$
- GCC analysis
  - Scalar evolution
  - Interprocedural analysis (e.g. constant propagation)
  - Create structured code
  - Alias analysis
- Extend polytope model?
  - Parametric strides $\rightarrow$ still too expensive, no optimized library

Space for improvements?

	Compile time			Hotspot		
Benchmark	**loops**	**conds**	**stmts**	**loops**	**conds**	**stmts**
base	4.66	1.52	2.90	9.45	7.68	10.61
SCEV_UNKNOWN	6.52	2.74	4.61	23.91	22.52	22.30
All conditions	5.08	18.04	18.50	5.58	17.39	19.58
All operands[1]	7.50	14.67	14.25	9.50	17.96	17.08
All structured control	65.78	40.52	48.43	66.70	57.57	63.65

Figure: Average coverage May 2009 [in %]

[1] is_simple_operand

The End